

Attorney Docket No.: 42390P10350

Express Mail No.: EL802887130US

UNITED STATES PATENT APPLICATION

FOR

**METHOD, APPARATUS, AND SYSTEM TO OPTIMIZE
FREQUENTLY EXECUTED CODE AND TO USE COMPILER
TRANSFORMATION AND HARDWARE SUPPORT TO HANDLE
INFREQUENTLY EXECUTED CODE**

Inventors:

Youfeng Wu
Li-Ling Chen

Prepared By:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Blvd., 7th Floor
Los Angeles, California 90025-1026
(714) 557-3800

BACKGROUND

(1) Field

The present invention relates to a method, apparatus, and system to optimize frequently executed code and to use compiler transformation and hardware support to handle infrequently executed code.

(2) General Background

Generally, the result of using a one-pass compiler is object code that executes much less efficiently than it might if more effort were expended in its compilation. Therefore, it is desirable to optimize object code or intermediate code that is translated into object code.

In an article entitled "rePlay: A Hardware Framework for Dynamic Program Optimization", CRHC Technical Report Draft, December 1999, by Sanjay J. Patel and Steven S. Lumetta, an optimization technique named "rePlay" is disclosed. However, rePlay relies mainly on hardware to form regions and optimize the regions at runtime. This reliance on hardware can be unrealistic since many optimizations can be complicated and require significant hardware and software compilation time.

Furthermore, other optimization techniques, including partial redundancy elimination (PRE) and partial dead-code elimination (PDE), can sometimes be ineffective and are quite complex to implement. PDE is disclosed in "Path profile guided partial dead code elimination using predication", Parallel Architectures and Compilation Techniques, 1997, by Rajiv Gupta, David E. Benson, and Jesse Z. Fang.

In addition, an optimization technique called "Superblock" is disclosed in "The Superblock: An Effective Technique for VLIW and Superscalar

Compilation", The Journal of Supercomputing, Kluwer Academic Publishers, 1993, pp. 229-248, by Wen-mei W. Hwu et al. Data and control flow for optimization and scheduling are generally simplified in a superblock. However, a superblock is still a multiple exit region. Thus the optimization and scheduling
5 need to handle issues such as side exit and speculation.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of an exemplary computing system in accordance with one embodiment of the present invention;

Figure 2 illustrates the result of a FastForward transformation in
5 accordance with one embodiment of the present invention;

Figure 3 generally outlines an exemplary process of constructing or forming FastForward regions in accordance with one embodiment of the present invention;

Figure 4 shows an example of updating the branch frequency;

Figure 5 shows an example of an FFR with multiple paths; and

10 Figure 6 shows an example of a general prevalent successor.

42390P10350

DETAILED DESCRIPTION

The present invention relates to a method, apparatus, and system to optimize frequently executed code and to use compiler transformation and hardware support to handle infrequently executed code.

- 5 Figure 1 is a block diagram of an exemplary computing system 100 in accordance with one embodiment of the present invention. Computing system 100 includes a central processing unit (CPU) 105 and memory 110 that is cooperatively connected to the CPU 105. CPU 105 can be used to execute a compiler 115 and a code optimizer 120, which are stored in the memory 110.
- 10 Compiler 115 is generally used to generate object code from a computer program written in a standard programming language. Compiler 115 includes a code optimizer 120 that is generally used to improve performance of the computer program. The store buffer 130 is cooperatively connected to the CPU 105 to assist the CPU 105 in running or executing the program speculatively.
- 15 To optimize code for a sequence of blocks with infrequent side exit branches in accordance with one embodiment of the present invention, the compiler 115 generally duplicates the code to form an initial FastForward region (FFR) and promotes the branch or check instructions in the FFR to ASSERT instructions. Branch promotion generally exploits the high frequency of
- 20 conditional branches that are strongly biased in one direction. When a strongly biased branch is detected, the branch is promoted into one with a static prediction. Since promoted branches generally require no dynamic prediction, the number of promoted branches allowed on a trace cache line is not limited by the bandwidth of the branch predictor. For more information on branch promotion, please see
- 25 the article entitled "Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing", In Proceedings of the 25th Annual International Symposium

on Computer Architecture, 1998, by Sanjay J. Patel, Marius Evers, and Yale N. Patt.

5 An ASSERT instruction can either take a predicate of an original branch or the source register of an original check as the operand. An ASSERT instruction typically fires when the predicate becomes false or the source register has a NAT value. As such, the FFR would typically have no side exit. It should be noted that compilers could typically optimize regions with no side exit much more effectively than regions with one or more side exits.

10 Compiler 115 then inserts a FastForward instruction at the beginning and a commit instruction at the end of the FFR. When the FastForward instruction of the FFR is executed, the address of the original code is saved and the rest of the FFR is executed speculatively. Store buffer 130 temporarily stores the results produced during the execution of a FastForward region. The results in the store buffer 130 are committed when the commit instruction is executed. However if 15 any of the ASSERT instructions in the FFR is fired, execution is rolled back to the original code; and the results in the store buffer 130 are discarded.

20 Figure 2 illustrates the result of a FastForward transformation in accordance with one embodiment of the present invention. A source code sample 205 is shown in (a). Exemplary original control flow graph (CFG) 210 corresponding to the exemplary source code 205 is shown in (b). Original CFG 210 includes two branches, "a==1" 215 and "a==2" 220, that are infrequently taken. As shown in (c), an FFR 250 is formed to include block B1' 225, block B3' 230, block B4' 235, block B6' 240, and block B7' 245. In the FFR, the two branches 215,220 of the original CFG 210 are converted into ASSERT instructions 255,260. 25 Instead of going to block B1 265 of the original CFG 210, control now goes to the FastForward instruction 275 in the FFR 250. The FastForward instruction 275 will record the checkpoint label 270. Checkpoint label 270 is the beginning address of

the original code, which is essentially the original code segment. If any of the ASSERT instructions in the FFR fires, the original code will be entered. If the commit instruction at the end of the FFR is reached and executed, the FFR execution has been successfully completed. At this time, the result in the store
5 buffer is updated to the state of the machine.

In performing the FFR transformation, a check instruction will be treated as a branch instruction. It should be noted that the ASSERT for a check instruction would fire on the NAT register value as well. Aside from the successor block connected by the cold edge, each candidate branch has only one successor
10 (referred to as a prevalent successor of the branch). A cold edge is generally a CFG edge with a very low probability of being taken. A candidate branch is generally a conditional branch instruction that has an outgoing edge that is a cold edge.

In addition, each candidate FFR is generally a single entry, single exit
15 region after all the cold edges are removed. A candidate FFR can have internal branches (including loop back branches), candidate branches, or a group of blocks that do not include any function calls and that are connected by unconditional branches. It should be noted that there are many types of code optimizations that are more effective when being applied to single entry, single exit regions than to
20 regions with more complex structures.

Accordingly, the general purpose of constructing or forming FastForward regions is to identify FFR regions having a high completion probability (i.e., a low probability of a side exit occurring), and having good opportunities for optimizations.

Figure 3 generally outlines an exemplary process 300 of constructing or
25 forming FastForward regions in accordance with one embodiment of the present invention. In block 305, standard available optimizations (including edge

profiling feedback and scheduling) are applied to a function to provide the best baseline code possible, prior to selecting and forming candidate FastForward regions for a function. Baseline code is generally code that does not include any FastForward regions. Furthermore, it should be noted that a FastForward region
5 is selected and formed only if the region can provide an improvement over the baseline code from which the region is derived.

In forming a FastForward region, a seed block for a candidate FFR is first selected (block 310). A typical seed block should be frequently executed, should include a candidate branch instruction, should not be already included in another
10 candidate FFR, and should not have any predecessor blocks that can be selected a seed block. Once it is selected, the seed block is duplicated and expanded or grown into an FFR (block 315). Following the duplication of the seed block, edge frequencies for the duplicated block and the original block are calculated. The duplicated seed block would serve as the initial head block and the initial tail
15 block of the current candidate FFR. After calculating the edge frequencies for the duplicated block, the tail block is examined.

If prevalent successor of the tail block is already in the current FFR, a back edge is being followed. The growth of the current FastForward Region can simply be stopped at the back edge. However for better performance, loop-peeling and
20 unrolling transformations can be used to grow an FFR along the back edges.

For example, if a loop has a small trip count, the loop can be peeled for a few iterations into the current FFR. It should be noted that loop peeling is implied when the region transformation is extended along loop back edges and will not stop until a loop exit block. If the loop has a small and constant number of
25 iterations, the loop can be completely unrolled along the hot path. A smaller edge frequency threshold value can be used to select the back edge since the first several iterations of the loop would more likely be taken than later iterations of the

loop. In addition, loop unrolling can be applied when a loop entry is encountered. To unroll a loop, the loop should have a single hot path and a prevalent post-exit block, which is a prevalent successor of the loop by treating the loop as a single node. In one embodiment, the process of loop unrolling can be performed using the logic represented in the following pseudo-code generally describing *Unroll_Loop()*.

```

Unroll_Loop(block, Candidate_FFR)
{
    duplicate a completely unrolled loop path into the current Candidate_FFR;
    return the prevalent post-exit block of the loop;
}

```

After loop unrolling and peeling, the edge frequency for the duplicated blocks is updated, and a new tail block for the FFR is selected to continue the growth of the FFR.

If the prevalent successor is not already in the FFR, the block is a candidate block to be added to the FFR. For each candidate block, certain operations are performed on the block, including duplicating the block, calculating the edge frequencies of the branch in the duplicated block, and making the duplicated block the new tail of the FFR. New candidate blocks would continue to be added (and the aforementioned operations would be performed on the newly added candidate blocks) until no additional blocks can be added to the FFR, or until the completion probability is lower than the threshold.

Once the growth of the FFR is stopped, the FFR can be trimmed by removing blocks near the head block or the tail block of the FFR (blocks 320-325). To trim a block near the head block of the FFR, each of the candidate branches is considered. The branch nearest to the head block would be considered first. For each candidate branch considered, a region is formed by adding the block containing the candidate branch to the tail of the FFR. Once the region is formed,

optimization and scheduling are applied to the region. During the application of the optimization and scheduling to the region, all candidate branches in the region are treated as ASSERTs. After optimization and scheduling are applied to the region, the average number of cycles (denoted T1) for the current FFR is
 5 computed. Also, the average number of cycles (denoted T) for the corresponding code in the original program is computed.

In one embodiment, the computation of T1 and T can be performed using the logic represented in the following pseudo-code generally describing *Compute_FFR_Cycles()*.

```

10  Compute_FFR_Cycles(list of block)
    {
      N = entry frequency of FFR;
      T1 = T = 0;
      completion_prob = 1;
15    R = overhead for the ASSERT to fire and branch to the original code
      for each ASSERT or candidate branch instruction in FFR
      begin
        t = cycles in blocks from head of FFR to the ASSERT or branch instruction;
        d = cycles in blocks from head of original code to corresponding branch;
20    K = firing frequency of the ASSERT (or branch frequency of the candidate branch);
        p = K/N;
        completion_prob -= p;
        T1 += (t + R + d) * p;
        T += d * p
25    end;
    ffr_commit_cycle = cycles from head to end of FFR;
    recovery_exit_cycle = cycles from head to end of the original code corresponding to the FFR;
    T1 += ffr_commit_cycle * completion_prob;
    T += recovery_exit_cycle * completion_prob;
30    return (T1, T);
  }

```

After T1 and T are computed, the benefit value of the candidate branch can then be calculated and saved. The benefit value of the candidate branch is
 35 essentially T minus T1 (T-T1). After all candidate branches are considered, the candidate branch with the best benefit can be identified. Furthermore, all blocks between the head block to the block before the identified candidate branch can be

discarded. However, if every benefit value of every candidate branch were not positive, the entire FFR would be discarded.

To trim blocks near the tail of the FFR, a similar process is adopted. The trimming of blocks near the tail of the FFR is shown in block 325 of Figure 3. For each candidate branch in the FFR, a region is formed from the head block to the prevalent successor of the candidate branch. After the region is formed, optimization and scheduling are applied to the region. During the application of optimization and scheduling to the region, all candidate branches in the region are treated as ASSERTS. After optimization and scheduling are applied to the region, the average number of cycles (denoted T1) for the current FFR is computed. Also, the average number of cycles (denoted T) for the corresponding code in the original program is computed.

After T1 and T are computed, the benefit value of the candidate branch can then be calculated and saved. The benefit value of the candidate branch is essentially $T - T1$ (T-T1). After all candidate branches are considered, the candidate branch with the best benefit can be identified. Furthermore, all blocks after the prevalent successor of the candidate branch can be discarded. However, if every benefit value of every candidate branch were not positive, the entire FFR would be discarded.

It should be noted that the blocks that are removed during the trimming of blocks near the head or tail of the FFR will be considered during the formation or construction of other FFR's.

During FFR formation or construction, the branch frequency needs to be updated for the original code to reflect the fact that some of the execution of the branches has been moved to the FFR (block 330). This branch frequency update is also needed during the repeated duplication of the same block into a candidate FFR. However, the edge frequency of the original program should not be

permanently changed during the formation or construction of the candidate FFR since the FFR may be trimmed or discarded at a later time. Therefore, the branch frequency for the original code is temporarily updated in a temporary area. After an FFR is finalized, the branch frequency for the original code needs to be permanently updated.

To update the branch frequency for the original code, the block frequency taken by blocks in the FFR is the subtracted or deducted from the block frequency in the original code. It should be noted that when an ASSERT fires, the blocks from the head to the branch in the FFR and the original code will both be executed. Therefore, the duplicated execution in the frequency must be considered and included. With the new block frequency information, branch frequency information can be updated. In one embodiment, the update block frequency can be computed using the logic represented in the following pseudo-code generally describing *Update_Freq_Prob()*, where *block_freq[b]* is the block frequency of block b, and *edge_prob[b₁, b₂]* is the branch probability on the edge from b₁ to b₂.

```

Update_Freq_Prob(Candidate_FFR)
{
    E = entry frequency;
    C = commit frequency;
    Candidate_FFR = {b1, b2, ..., bn} whose corresponding original blocks are {a1, a2, ..., an};
    for each block bi, i=1,...,n, in Candidate_FFR
        begin
            find the corresponding recovery block (ai) in original code;
            block_freq[bi] = E * edge_prob[a1, a2] * ... * edge_prob[ai-1, ai];
            // the original block needs to take ASSERT frequency into consideration
            block_freq[ai] = block_freq[ai] - C
        end;
    for each block bi, i=1,...,n-1, in Candidate_FFR
        begin
            for each successor s of bi
                edge_prob[bi, s] = block_freq[s] / block_freq[bi]
            end;
            for each recovery block ai, i=1,...,n-1, in Candidate_FFR
                begin
                    for each successor s of ai
                        edge_prob[ai, s] = block_freq[s] / block_freq[ai]
                    end;
                end;
            end;

```

}

Figure 4 shows an example of updating the branch frequency. Initially, B1' 405 has frequency of 100. The probability of ASSERT a==1 is 1/100; and the number of ASSERT in B1' is 1. Therefore, 1 is added back to B1. The ASSERT probability for ASSERT a==2 is 10/200. Thus the ASSERT frequency is 4.95 (i.e., 99×0.95), which will be added back to B1 410, B3 415, and B4 420. Also, the FFR has an entry frequency of 100 and a commit frequency of 94.05. As a result, the block will have the correct frequency with ASSERT overhead taken into account and be possibly incorporated into multiple FastForward groups.

In one embodiment, the process of identifying a candidate FFR can be performed using the logic represented in the following pseudo-code generally describing *Construct_FFR()*, which invokes *Compute_Benefit()*, *Update_Freq_Prob()*, and *Loop_Unrolling_Satisfied()*, where *tmp_freq* is an array for storing block frequency information temporarily.

```
Construct_FFR(block, Candidate_FFR)
{
    Queue = all the hot blocks with a candidate branch and without a predecessor with a candidate branch;
    while (block = DeQueue(Queue))
    { tmp_freq[ ] = copy of all block frequency of the function;
      Candidate_FFR = new_FFR();
      duplicate_blk = duplicated block of the original block;
      tmp_freq[duplicate_blk] = tmp_freq[block];
      Add block to Candidate_FFR;
      tmp_freq[block] = 0 //initially for a seed block

      //determining prevalent successor using the tmp_freq[ ] information
      while (block has a prevalent successor succ_blk)
      {
          if (succ_blk is in another Candidate_FFR)
              break;
          if (Loop_Unrolling_Satisfied(succ_blk))
              succ_blk = Unroll_Loop(succ_blk, Candidate_FFR)
          else begin
              duplicated_succ = duplicated succ_blk;

              //computing block frequency
              tmp_freq[duplicated_succ] = tmp_freq[duplicate_blk] * edge_prob[block, succ_blk];
```

```

    tmp_freq[succ_blk] = tmp_freq[succ_blk] - tmp_freq[duplicated_succ]
    end;
    blk = succ_blk;
  } //while (block has a prevalent successor succ_blk)
5  } // while (block = DeQueue(Queue))

```

L1:

```

10  //Trimming near the beginning
    main_head = head of Candidate_FFR
    best_head = NULL;
    best_benefit = 0;
    for each candidate branch from the one nearest to the head block of Candidate_FFR
    begin
15    blk = the block containing the candidate branch;
L2:    benefit = Compute_Benefit(blocks from blk to end of the Candidate_FFR);
        if (benefit > best_benefit) begin
            best_head = blk;
            best_benefit = benefit;
20        end;
    end;

    if (best_benefit <= 0) begin
25    EnQueue(head->prevalent_successor, Queue);
        remove Candidate_FFR;
        continue;
    end;

    if (best_head != main_head) begin
30    new_seed = main_head;
        remove blocks from main_head to the block before the best_head from FFR;
        Candidate_FFR->head = best_head;
        if (new_seed has a candidate branch)
            EnQueue(new_seed, Queue)
35    end;

```

```

    //Trimming near the end
    best_tail = NULL;
    best_benefit = 0;
    for each candidate branch of Candidate_FFR
    begin
        succ_blk = the prevalent successor of the candidate branch;
40    L3:    benefit = Compute_Benefit(blocks from head to succ_blk);
        if (benefit > best_benefit) begin
            best_tail = succ_blk;
            best_benefit = benefit;
            end;
50    end;

    if (best_benefit <= 0) begin
        EnQueue(head->prevalent_successor, Queue);
        remove Candidate_FFR;

```

```

        continue;
    end;

    if (best_tail != tail of FFR) begin
5      new_seed = prevalent successor of best_tail;
      remove blocks from new_seed to the tail of Candidate_FFR;
      Candidate_FFR->tail = best_tail;
      if (new_seed has a candidate branch)
10        EnQueue(new_seed, Queue)
    end;

    // Update block frequency and edge probability for the finalized FFR and corresponding recovery code
    Update_Freq_Prob(Candidate_FFR);

15    for each candidate branch in Candidate_FFR
        begin
            convert the candidate branch to an ASSERT;
        end;

20    create fastforward and commit instructions and connect Candidate_FFR to original code;
} //end of Construct_FFR()

25    Compute_Benefit(list of blocks)
    {
        treat the list of block as a region;
        treat all branches as asserts;
        schedule and optimize the region;
30    identify the corresponding original_region;
        (T1, T) = Compute_FFR_Cycles(region, original_region);
        return (T-T1);
    } //end of Compute_Benefit()

35    Loop_Unrolling_Satisfied(block)
    {
        if (block is a loop entry
            && block->loop has a small, fixed number of iterations and small hot path loop body
40        && block->loop has a prevalent successor block)
            return TRUE;
        else return FALSE;
    } //end of Loop_Unrolling_Satisfied()

```

45 It should be noted that the above pseudo-code generally describing Construct_FFR() includes labels L1, L2, and L3. The reason for including these labels will be more apparent as will be shown below.

As shown above, optimizations and scheduling of a sub-FFR are repeatedly performed to determine the benefit for identifying the best FFR. The repeated optimization and scheduling is needed since the trimming of any block will significantly affect the schedule of the remaining region. However, the repeated optimization and scheduling will also increase the compilation time. Therefore, an alternative region formation technique could be employed to trade off the compilation time and the optimality of the resulting FFR. In the alternative technique to construct or form FFR regions, the optimization and scheduling of each candidate FFR is performed only once.

- 10 In the alternative technique to construct or form FFR regions where the optimization and scheduling of each candidate FFR is performed only once, the region-based optimization and scheduling is applied to the whole candidate FFR before trimming any blocks. For each candidate branch in the FFR, a region is formed from the head block to the prevalent successor of the candidate branch.
- 15 After that, the average number of cycles (denoted T1) for the current FFR is computed. Also, the average number of cycles (denoted T) for the corresponding code in the original program is computed. In one embodiment, the computation of T1 and T can be performed using the logic represented in the pseudo-code generally describing *Compute_FFR_Cycles()*. After T1 and T are computed the
- 20 benefit value of the candidate branch can then be calculated and saved. The benefit value of the candidate branch is essentially $T - T1$.

- To trim blocks near the head block of the FFR, each of the candidate branches from the block nearest to the head block is considered. The candidate branch with the best benefit value is identified. Then, all blocks from the head
- 25 block to the predecessor block of the identified candidate branch are discarded. If no candidate branch has a positive benefit value, the entire FFR will be discarded.

To trim blocks near the tail of the FFR, the candidate branch with the best benefit value is identified. Then, all blocks after the prevalent successor of the identified candidate branch are discarded. If no candidate branch has a positive benefit value, the entire FFR will be discarded.

5 In one embodiment, the alternative region construction or formation technique, in which the optimization and scheduling of each candidate FFR is performed only once, can be performed using logic similar to the above pseudo-code generally describing *Construct_FFR()* with modifications at labels L1, L2, and L3 as follows:

10 • A statement to schedule and optimize the Candidate_FFR should be added at label L1.

• The statement at label L2 should be replaced with the following pseudo-code:

benefit = Compute_Simple_Benefit(blocks from head to succ_blk)

• The statement at label L3 should be replaced with the following pseudo-code:

15 *benefit = Compute_Simple_Benefit(blocks from head to succ_blk)*

In one embodiment, the logic of *Compute_Simple_Benefit()* can be generally described in the following pseudo-code:

20 *Compute_Simple_Benefit(list of blocks)*
 {
 treat the list of block as a region;
 treat all branches as asserts;
 identify the corresponding original_region;
 (T1, T) = Compute_FFR_Cycles(region, original_region);
 25 *return (T-T1);*
 } *//end of Compute_Simple_Benefit()*

After a candidate FFR is constructed or formed, candidate branches are converted to ASSERT instructions. However if a candidate branch has both of its successors inside the same FFR, the candidate branch will not be converted to an

ASSERT instruction. Furthermore, the original code corresponding to the candidate FFR is connected with FastForward and commit instructions in the candidate FFR.

Next, optimizations and instruction scheduling are performed to the whole function. The optimizations and instruction scheduling should keep the semantics of the FFR the same as its corresponding original code. One simple method is to keep the semantics of each FFR unchanged. For example, optimizations that may change the semantics of the FFR (e.g., code hoisting) should first be applied to the inside of the candidate FFR; and each region should then be treated as an atomic operation when optimizations are applied globally. It should be noted that many global optimizations, such as copy propagation, dead code elimination, etc. do not change the semantics of the FFR so they can be applied across FFR boundaries.

It should be noted that the aforementioned techniques to form regions could construct FastForward regions with multiple paths. Figure 5 shows an example of a multi-path FFR 500. In the figure, the edge B1->B4 505 is included in the FFR 500 as both successors of B1 510 are in the same FFR 500.

Furthermore, the aforementioned techniques to form or construct regions could be extended to allow more general multiple-path FFR by extending the concept of prevalent successors. Figure 6 shows an example of a general prevalent successor. The prevalent successor S 605 of a block B 610 is a block that B 610 reaches with a high probability, possibly going through a Directed Acyclic Graph (DAG) 615. It should be noted that no block between B and S can be a prevalent successor of B. A special case of prevalent successor is when B reaches S directly. In Figure 5, block B1 510 reaches block B4 515 and block B7 520 with a probability of 1.0; however, only block B4 515 is the prevalent successor of block B1 510.

With the generalized concept of prevalent successor, the region formation algorithm can be used to form general DAG FastForward regions. It should be

noted that the FFR should still be a single entry and single exit region. Any branch in the FFR that jumps to the outside of the FFR will be converted to an ASSERT.

5 The FastForward technique can be applied in a single-threaded or multi-threaded execution models. The techniques presented earlier are based on a single-threaded model. The original code generally does not need to be executed most of the time if the FFR execution is committed successfully. In addition, the probability of hitting an ASSERT is low. Once an ASSERT is fired, the original code will be started from the beginning, and the result of the FFR execution in the hardware store buffer will be discarded.

10 In a multi-threaded model, both the original code and FFR will be executed simultaneously. FastForward region is executed speculatively. If none of the ASSERT instructions inside the FFR were fired, the speculative thread may commit and act as a main thread, assuming that FFR is highly optimized and will finish earlier. If any of the ASSERT instructions fires, the speculative thread dies and the main thread just continues. It is not necessary to start the original code after hitting an ASSERT. Therefore, the execution time of recovery overhead is almost none as the original code started simultaneously. On the other hand, the hardware support on multi-threading synchronization may require more complicated design than the single-threaded FastForward store buffering.

20 It should be noted that functional components, as shown in the figures and described above in the text accompanying the figures, could be implemented using software code segments. If the aforementioned functional components are implemented using software code segments, these code segments can be stored on a machine-readable medium, such as floppy disk, hard drive, CD-ROM, DVD, tape, memory, or any storage device that is accessible by a computing machine.

While certain exemplary embodiments have been described and shown in accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific constructions and arrangements shown and
5 described, since various other modifications may occur to those ordinarily skilled in the art.

42390P10350